Microprocessors and Microsystems 60 (2018) 86–95

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Reconfiguration Control Networks for FPGA-based TMR systems with modular error recovery

Nguyen T.H. Nguyen[*,a,b], Dimitris Agiakatsikas[a], Zhuoran Zhao[a], Tong Wu[a], Ediz Cetin[c], Oliver Diessel[a], Lingkan Gong[a]

[a] School of Computer Science and Engineering, UNSW Sydney, Australia
[b] Faculty of Computer Science and Engineering, HCMC University of Technology, Vietnam
[c] School of Engineering, Macquarie University, Australia

ARTICLE INFO

Keywords:
Reconfiguration Control Networks
Dynamic Partial Reconfiguration
SRAM-based FPGA
Fault injection
Radiation effects
Scrubbing
Single Event Upsets
Triple Modular Redundancy
Reliability

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) provide ideal platforms for meeting the computational requirements of future space-based processing systems. However, FPGAs are susceptible to radiation-induced *Single Event Upsets* (SEUs). Techniques for dynamically reconfiguring corrupted modules of *Triple Modular Redundant*(TMR) components are well known. However, most of these techniques utilize resources that are themselves susceptible to SEUs to transfer reconfiguration requests from the TMR voters to a central reconfiguration controller. This paper evaluates the impact of these *Reconfiguration Control Networks* (RCNs) on the system's reliability and performance. We provide an overview of RCNs reported in the literature and compare them in terms of dependability, scalability and performance. Most importantly, we compare the performance of soft networks with that of a hard network that utilizes the *Internal Configuration Access Port*(ICAP) available in advanced Xilinx devices to periodically read the TMR voter states. We have implemented our designs on a Xilinx Artix-7 FPGA to assess the resulting resource utilization and performance as well as to evaluate their soft error vulnerability using analytical and fault injection techniques. Results show that, of the RCN topologies studied, the ICAP-based approach is the most reliable despite having the highest network latency. We also conclude that a module-based recovery approach is less reliable than scrubbing unless the RCN is implemented with redundancy and repaired when it suffers from configuration memory errors.

1. Introduction

In order to reduce mass and power consumption and to achieve desired processing performance, space missions are increasingly integrating a number of applications, e.g., flight control, signal and image processing, data compression and encryption, within a single SRAM-based FPGA. However, a major challenge for deploying such FPGAs in space, particularly if they are commercial, off-the-shelf devices, is to mitigate radiation-induced soft errors, primarily *Single Event Upsets* (SEUs). Soft errors can alter the contents of user memory elements, such as flip-flops and block RAM as well as the SRAM configuration memory, thereby producing both transient and permanent circuit malfunctions [1,2].

A popular SEU mitigation approach is *Triple Modular Redundancy* (TMR) [3,4]. TMR involves the triplication of the user circuit and the use of majority voters to detect and mask SEUs affecting one of the three functionally equivalent modules. TMR raises system reliability by masking errors that affect both the implemented circuit as well as the underlying configuration memory. However, TMR is unable to correct errors that are trapped within a cyclic user circuit, or within the configuration memory. Errors trapped in user circuitry can be corrected by resetting the faulty module or by resynchronizing the module with its siblings. To deal with configuration memory errors, TMR is usually combined with error recovery techniques, such as with scrubbing [2,5], or with modular reconfiguration [6–8].

Both scrubbing and modular reconfiguration rely on *Dynamic Partial Reconfiguration* (DPR) to correct configuration memory errors. Scrubbing is typically initiated periodically and commonly involves reading back each configuration memory frame, checking for errors, correcting any that are found and writing back the corrected frame, when necessary. In contrast, modular reconfiguration is commonly triggered when repeated errors are detected by the voter associated

* Corresponding author at: School of Computer Science and Engineering, UNSW Sydney, Australia.
E-mail addresses: h.nguyentran@student.unsw.edu.au, nthnguyen@hcmut.edu.vn (N.T.H. Nguyen), d.agiakatsikas@student.unsw.edu.au (D. Agiakatsikas), zhuoran.zhao@student.unsw.edu.au (Z. Zhao), tong.wu@student.unsw.edu.au (T. Wu), ediz.cetin@mq.edu.au (E. Cetin), o.diessel@unsw.edu.au (O. Diessel), lingkang@cse.unsw.edu.au (L. Gong).

https://doi.org/10.1016/j.micpro.2018.04.006
Received 31 May 2017; Received in revised form 9 April 2018; Accepted 16 April 2018
Available online 17 April 2018

with a TMR component and involves rewriting the configuration memory for the module that has been found to be in error. Scrubbing, which could be referred to as a *frame-based* recovery technique, is thus more fine-grained in its corrective action but involves reading or writing the entire configuration memory contents On the other hand, *module-based* recovery methods are more coarse-grained in that the configuration memory contents of a complete module are rewritten. Multiple configuration memory errors affecting the one module can thus be corrected in a single action and correction is typically completed more promptly.

Scrubbing consumes more energy than modular reconfiguration [9] because it is invoked periodically rather than when errors are present. Scrubbing also has a higher *Mean Time To Detect* (MTTD) errors than modular reconfiguration. While modular reconfiguration is triggered by a voter signalling the presence of an error, state-of-the-art FPGAs may include on the order of a billion configuration bits and the time required to read back the entire configuration memory during a scrub cycle may exceed 120 ms at least. This means that SEUs will be detected in the system after 60 ms on average, which could be too large for critical systems.

Modern FPGAs, such as those offered by Xilinx, include built-in resources and provide soft IP to aid scrubbing [10], but use of a module-based error recovery approach is not supported nearly as well and design complexity is noticeably increased. For example, a designer who wishes to employ module-based error recovery needs to design a suitable reconfiguration controller, a method for signaling reconfiguration requests, storage for partial bitstreams and a data resynchronization mechanism.

In this paper, we focus on the choice and design of a *Reconfiguration Control Network* (RCN), which is an infrastructural component that collects reconfiguration requests from the system's TMR voters and communicates these to an internal or external reconfiguration controller [1,2]. The performance and reliability of the RCN is important for a number of reasons. On the one hand, the latency of the RCN has a direct impact on the *Mean Time To Detect* (MTTD) errors in the system. On the other hand, the RCN is often implemented in a non-redundant manner and therefore introduces a single point of failure that can easily compromise system reliability. It is also possible to trigger a scrub cycle using an RCN to collect status messages from the voters of TMR components rather than relying on a periodic or error rate-based trigger. Hence the results presented in this paper are of relevance to any SRAM FPGA-based TMR system irrespective of the type of error recovery method used.

Considering the importance of the RCN in module-based recovery mechanisms, we provide a comprehensive study of the variety of RCNs mentioned in the literature. The RCN is typically realized by utilizing configurable resources such as *Configurable Logic Blocks* (CLBs) and programmable interconnection resources. However, an RCN can also be implemented using the FPGA's hardwired configuration network. In this work we distinguish between RCNs that are implemented in programmable logic, which we refer to as *soft* networks, and those that are implemented using hardwired non-programmable resources, which we refer to as *hard* networks.

Soft RCNs can be realized with simple star networks [6,7] or with more complex networks such as bus networks [11] and token ring networks [8]. In soft networks, routing congestion may occur as the number of TMR components in the system increases. In contrast, hard networks [12] rely on the built-in configuration network of the FPGA to provide access to the state of health of TMR components. This results in reduced demand for routing resources.

In this paper, we compare four RCNs with respect to reliability, latency, scalability and power consumption. Fault injection experiments are conducted to evaluate the impact of each RCN on system reliability. We demonstrate that the hard network, which uses the *Internal Configuration Access Port* (ICAP) to read the voter state, achieves the highest reliability in a case study that is implemented on the RUSH (Rapid recovery from SEUs in Reconfigurable Hardware) payload [13]. We also show that MTTD is greatest for the ICAP-based approach due to the relatively large latency involved in retrieving user state this way but demonstrate an effective optimization that significantly narrows the gap between this hard approach and the soft RCNs. Finally, we assess the reliability of a real system employing module-based recovery relative to the same system using blind scrubbing. We have determined that scrub-based error recovery results in higher reliability unless the RCN is itself triplicated and repaired when its configuration becomes corrupted.

The paper is organized as follows: Section 2 provides an overview of TMR systems that employ modular reconfiguration for configuration memory error recovery. Section 3 reviews the literature available on RCN designs, with Section 4 describing the architecture of the various RCN types we studied. Sections 5–7 provide background on the frame readback technique employed by the ICAP, the fault emulation system we implemented to assess the soft error vulnerability of our designs and the model we used to evaluate the reliability of our implementations. Section 8 describes our experimental method and reports our findings while concluding remarks and directions for further study are given in Section 9.

This paper extends the previous work [14] in the following ways:

– We provide a background on common error recovery techniques such as scrubbing and MER that have been described in the literature (Section 1).
– We provide an overview of TMR systems using modular error recovery (MER) and describe the advantages and disadvantages of MER compared with scrubbing (Section 2).
– We detail the RCN topologies that have been mentioned in the literature and that are studied in our evaluation. We particularly focus on use of the Internal Configuration Access Port (ICAP) that available in advanced Xilinx FPGAs to readback configuration frames that contain information regarding the health of the systems TMR components (Sections 4 and 5).
– We describe details of our fault injection experiment, which is completely novel in terms of how to stimulate the inputs of the Design Under Test (DUT) (Section 6).

2. An overview of TMR with module-based configuration memory error recovery

SRAM-based FPGAs can be considered to be devices that consist of two layers, namely an application layer and a configuration layer. The former is composed of all programmable logic and memory resources that are used to implement user designs while the latter comprises the configuration memory and the logic needed to access this memory. User designs are configured by means of the configuration memory bits. *Module-based configuration memory Error Recovery* (MER) takes advantage of this to allow for the run-time modification of a user design by loading a partial configuration file [15]. While a TMR component in the system is running, a partial configuration file can be loaded to overwrite a reconfigurable module without compromising the integrity of its two sibling modules. Module-based partial reconfiguration can therefore be combined with TMR to alleviate SEUs that have affected configuration memory. This is because the voters associated with TMR components can localize faulty areas, while the use of modular reconfiguration can swiftly clean these errors [6].

Fig. 1 illustrates an FPGA-based TMR system that employs MER. The *voter* associated with each TMR component identifies which module, if any, is suffering from a persistent fault, and raises a reconfiguration request. Requests from the voters of different TMR components across the device are transmitted through a Reconfiguration Control Network (RCN). The RCN identifies which module needs to be reconfigured and sends identifying information to a *Reconfiguration Controller* (RC). The RC fetches the corresponding partial bitstream from *off-chip memory*

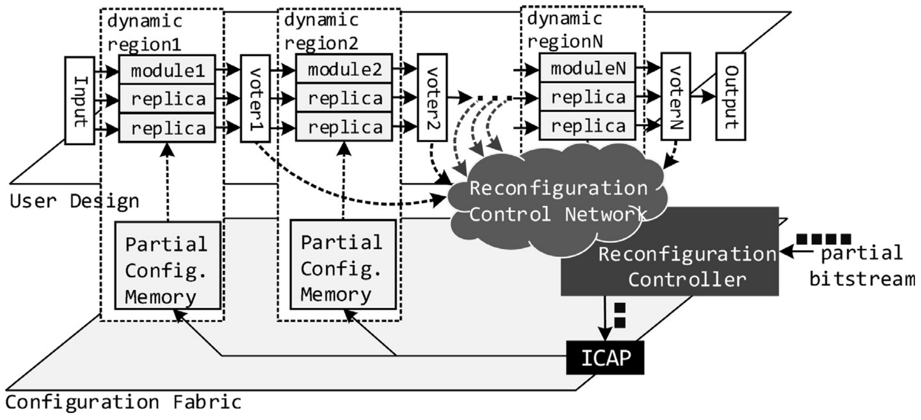

Fig. 1. An example TMR with MER system diagram.

and reconfigures the faulty module via the *configuration fabric* which is accessed through the ICAP. (We use the terminology used by Xilinx for their partially reconfigurable devices, however, the concepts we describe are equally applicable to any partially reconfigurable devices, such as the Stratix 10 family, available from Altera.) After the faulty module has been reconfigured and resynchronized with the remaining two modules of the TMR component, the voter resumes its normal operation.

As mentioned in Section 1, the use of MER with TMR affords several benefits over scrubbing (frame-based error recovery), but also introduces several additional costs. The main benefits of MER are that the system is able to respond more dynamically to configuration memory errors.

Configuration memory errors are detected by individual voters when faults occur repeatedly within the one module. Rather than having to read back half of the FPGAs configuration frames on average to detect an error when incorporating scrubbing recovery [16], MER utilizes the voters of the system to rapidly detect errors within a few tens to hundreds of clock cycles. When an error is detected by a voter, a reconfiguration request that identifies the module to be reconfigured must be conveyed to the RC. The performance of the RCN determines the latency in transmitting this request. As the investigation of this paper indicates, this is typically possible within a few microseconds at most. Overall, the time to detect errors with MER is therefore in the order of microseconds, compared to milliseconds for scrubbing.

Repairing errors by MER is also typically much faster than for scrubbing since MER is usually achieved by reconfiguring the faulty module, whereas scrubbing necessitates reading the entire memory contents of the device to check the CRC or writing the full device configuration. Thus the size of the module to be repaired, rather than the device size, determines the correction time. Most work focuses on modules of the size of a small number of device resources (a few CLBs, FFs and routing) to medium-sized components such as linear filters, a processor stage, etc. Typical module sizes range between 10s and 100s of configuration memory frames (several kB) as opposed to 10,000s of frames (several MB) available at device level. It can therefore be expected that error repair also takes two to three orders of magnitude less time for MER. This responsiveness to errors afforded by MER can be crucial to the success of a mission when errors occur in rapid succession at critical stages such as during spacecraft maneuvers, communications with Earth and during remote sensing when particular areas of interest are observed.

Since MER replaces all the configuration memory contents for a module, it inherently repairs multiple errors such as *Multiple Bit Upsets* (MBUs), which simple scrubbing modes, such as *Single Error Correction, Double Error Detection* (SECDED) [10], are generally unable to fix. Only more complex methods of scrubbing, such as frame replacement, which comes with many of the costs of MER, are capable of dealing with MBUs efficiently [10].

MER comes with some considerable costs, however, most of these have not yet been quantified as they mainly deal with greater design complexity. The designer needs to consider modifications to the voter design to be able to detect configuration memory errors; an RCN is needed to convey reconfiguration requests to the RC; an RC that can process reconfiguration requests and manage modular reconfiguration needs to be designed; secure storage for the module-based partial bitstreams needs to be incorporated; a module-based resynchronization method also needs to be developed. If the designer were to choose a simple scrubbing-based configuration memory recovery method instead, most of these design modifications are not needed. Instead, the designer could make use of IP provided by the device vendor and inbuilt hardware to perform the scrubbing. This is the case when SECDED is used [10]. If a frame replacement scrubbing method is used, then the designer also needs to develop a more sophisticated scrub controller, implement a method for securely storing the device configuration, and implement a method for accessing the golden configuration on a frame-by-frame basis.

Of considerable concern is that much of the additional logic used to implement and support MER may be implemented in a non-redundant manner and therefore introduces additional single points of failure. In this paper, we consider the impact of the RCN on system reliability. In this work we focus on non-redundant RCN topologies. This provides a base-line from which greater reliability could be achieved if the RCN itself were triplicated. Nevertheless, irrespective of the configuration memory error recovery approach taken, FPGA-based TMR systems inevitably include non-redundant components such as clock managers, ICAP, off-chip ports, etc. Periodically invoked recovery, such as scrubbing, is likely to deal with configuration memory errors that occur in these components better than MER does, since non-redundant components don't have voters to trigger MER.

3. Related work

Several types of networks for aggregating reconfiguration requests from TMR voters have been described in the literature. These include examples of a star network [6,7], a bus network [11], a token ring network [8], and an ICAP-based readback approach [12].

Star networks use simple interfaces to connect the voter outputs, which are distributed across the device, to a central *Network Controller* (NC) [6,7]. In star networks, the interconnecting wires may need to span the entire device and therefore pass through numerous programmable interconnection resources. This not only increases their susceptibility to SEUs, but also introduces latency. Star networks described in the literature typically involve polling of the remote (voter) interfaces by a central controller implementing a round-robin algorithm. It would be feasible to also consider an interrupt-driven approach whereby voters interrupt a central arbiter to transfer a reconfiguration request.

In [11] the authors utilized the *Advanced eXtensible Interface* (AXI) core to transfer the outputs of individual modules to a central voter. While not serving as an RCN, any shared messaging resource, such as this bus, could be used to convey reconfiguration requests from distributed components to a central controller. The use of a shared bus allows new modules to be readily integrated into the system while avoiding the use of the dedicated routing resources found in star networks. However, a bus requires more complex interconnection interfaces than star networks, which results in an increased soft-error vulnerability, power consumption and latency.

In [8] a token ring network is implemented that spans all voters in a daisy-chained manner. The design uses complex network interfaces that require significantly more logic than the endpoints of the point-to-point connections found in the star networks in [6]. However, token ring topologies usually link neighbouring components and therefore utilize a reduced number of global wires for interconnecting them. In contrast, star and bus topologies realize mixed distance connections and thus utilize various interconnection resources, including both local and global wires. Usually, SRAM-based FPGAs integrate more local than global wires and therefore token ring networks, which tend to utilize more local wires, are considered to be more scalable than star and bus networks. However, in token ring networks, the latency increases with the number of components on the network. A drawback of this topology is that when a link suffers a configuration memory error the ring no longer functions as intended, whereas the star topology is inherently more robust as all links are independent.

A fourth approach that has been described in the literature makes use of the ICAP to read the outputs of the TMR modules or to read the voter states in a round-robin fashion [12]. The former uses software to centrally compare the outputs of each module in order to reduce the overheads of distributed voting and to reduce the likelihood of the voter mechanism becoming corrupted. An ICAP-based communications scheme eliminates the need for a soft network and therefore reduces routing pressure, implementation time, and improves reliability. Reliability is enhanced since the built-in hard reconfiguration network is utilized to obtain module or voter outputs. This approach has the potential to be scalable as it does not require user routing resources and utilizes a moderate amount of logic to implement the central controller.

4. RCN architectures

In this section, we describe the architecture of the four RCN types found in the literature and provide average latencies for obtaining a reconfiguration request.

4.1. RCN types

Each RCN is composed of distributed *Network Interfaces* (NIs), a central *Network Controller* (NC) and an *interconnection network* between them as illustrated in Fig. 2. In the figure, each majority voter provides a 2-bit *error info* signal to the corresponding NI. Three possible values of the *error info* signal — 00, 01, and 10 — represent the error states of the triplicated modules respectively, while the value 11 indicates there is no error. Note that a voter raises a *Reconfiguration Request* (RR) to the NC by means of the *error info* value. Once the NC receives an RR, it issues a *req* and a $(log_2 N + 2)$-bit *module id* signal to the *Reconfiguration Controller* (RC), which invokes the module-based error recovery (MER) process to recover the faulty module. After MER is completed, the RC asserts a *done* signal to the NC.

4.1.1. Star network

Fig. 3(a) illustrates at an overview level a typical star network implementation. Each NI contains a 2-bit buffer that connects directly to the NC. The NC consists of two modules, namely an arbiter and a multiplexer. The arbiter selects which voter is to be checked in a round robin manner while the multiplexer transfers the *error info* from the selected voter to the arbiter. When the arbiter receives an RR, it sends *req* and *module id* signals to the RC. The RC invokes modular reconfiguration for the faulty module before issuing a reconfiguration *done* signal to the arbiter for it to resume voter checking.

4.1.2. Bus network

We detail a simple bus network that aggregates the error message from the voters as illustrated in Fig. 3(b). The bus network works similarly to the star network. The multiplexer architecture of the star network is replaced with a global address signal. The registers in the NI are controlled by the *address decoder* (dec) and the register outputs drive the common bus signal through an *OR* gate. If the bus signal indicates an RR, the NC triggers the reconfiguration process in the RC.

4.1.3. Token ring network

Fig. 4 shows the basic token ring network architecture. In contrast to the star architecture, the NIs provide a $(log_2 N + 2)$-bit message that indicates which module is in error. When the *val* signal toggles and the message on the data signal is a token, the *OutRegs* in the NI latch the token before passing it on to downstream NIs. If an NI needs to send an RR, it keeps the token and its *OutRegs* latch the corresponding *module id*. The downstream NIs pass this message to the NC. The NC operates in a similar manner to that described for the star network. When a modular reconfiguration has occurred, a *done* message is released to signal to the requesting NI that the reconfiguration has been completed and to release the token.

4.1.4. ICAP-based voter checking

The ICAP-based readback approach eliminates the soft interconnection network by using a modified RC component to check on the TMR component voters directly. The voter *error info* signal is registered and the RC polls these register outputs in a round-robin manner via ICAP readbacks. The method is further detailed in Section 5. When the RC receives a reconfiguration request, it triggers a reconfiguration operation to correct the faulty module.

4.2. RCN latency

RCN latency is defined as the average period of time needed for the NC to receive a reconfiguration request from a voter. As described in Section 4.1, all four networks check voters in a round-robin manner. Thus, assuming a system with N TMR components or NIs and one NC, the average latency of the token ring network is given by

$$latency = (N + 1) \times c_{hop} \times \frac{1}{F_{network}}, \qquad (1)$$

where c_{hop} denotes the number of clock cycles per node hop, and $F_{network}$ denotes the maximum clock frequency of the RCN. Eq. (1)corresponds to the average time needed for the token to arrive (half the ring) and the time for the request to make it back to the NC (also half the ring).

The RCN latency for all other topologies is given by

$$latency = \frac{N}{2} \times c_{hop} \times \frac{1}{F_{network}}, \qquad (2)$$

which corresponds to the time it takes to check half the voters in the system before the one that wishes to raise a reconfiguration request is checked.

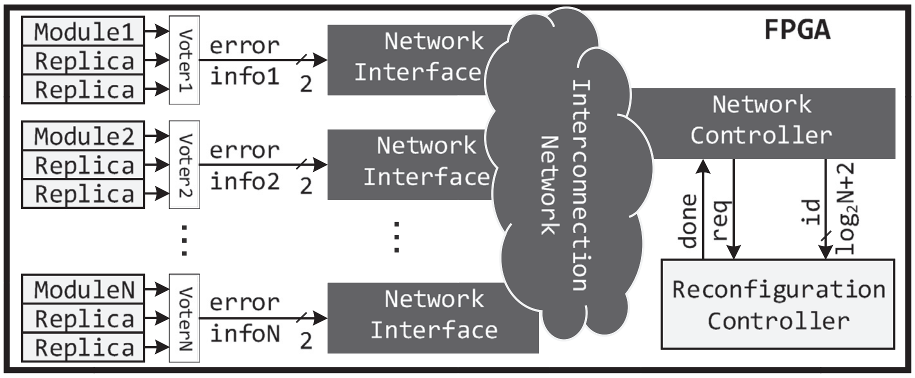

Fig. 2. Components of an RCN.

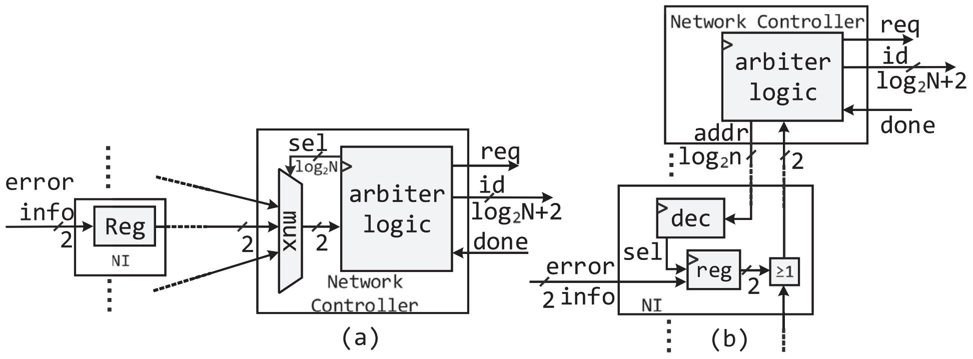

Fig. 3. The architecture of a star network (a) and of a bus network (b).

5. Configuration frame readback

The Xilinx FPGA configuration memory is arranged in frames that are tiled about the device. These frames are the smallest addressable segment of the configuration memory space. The frame size varies among FPGA families; in the case of Xilinx 7–Series FPGAs, it consists of 101 32-bit words. Each frame possesses a unique address that can be used to dynamically read or write to the configuration memory.

Xilinx 4—7 Series devices allow users to read the configuration memory via the ICAP. There are two modes of readback, namely *Readback Verify* (RbV) and *Readback Capture* (RbC) [15]. We use the RbC mode to check the voter state of each TMR component since this mode allows the state of the CLB configuration memory cells to be read. This can be done by issuing a GCAPTURE command to the ICAP so as to sample all CLB register values into configuration memory cells. These values can then be read back along with the configuration frame containing the voter status bits. However, designers must know the frame address and configuration bit offset of the SRAM cell corresponding to the desired output of the voter for this approach to work. These parameters are given in the logic allocation (`*.ll`) file, which is automatically generated by the Xilinx ISE/Vivado design tools. The logic allocation file includes four fields, namely a *bit offset*, a *frame address*, a *frame offset*, and *information* for each configured resource as depicted in Fig. 5. The registers corresponding to the voter status of a TMR component are determined from the information fields that then allow the frame addresses and frame offsets to be extracted.

Xilinx devices expect a specific sequence of commands to be sent to the ICAP in order to read a data frame [15]. A frame read request necessitates the read of a dummy word and a pad frame before the desired data frame can be read. The time to read a frame in Xilinx 7–Series FPGAs is approximately 230 clock cycles. This includes 20 clock cycles for issuing initialization commands, 203 clock cycles for the frame read, and 10 clock cycles for issuing concluding commands [15]. The frame read time depends on the throughput of the ICAP, which supports 32-bit transfers at a rate of 100 MHz. At the maximum rated speed, the time for reading a frame through the ICAP primitive in 7–Series FPGAs is thus approximately 2.3 us. This latency can be reduced if voter registers are placed at the bottom of each clock region so that the voter registers are located at the beginning of the data frame. When this is done, the frame read can be aborted after the voter registers have been read. The frame read time can thus be reduced by as much as 1 us [15].

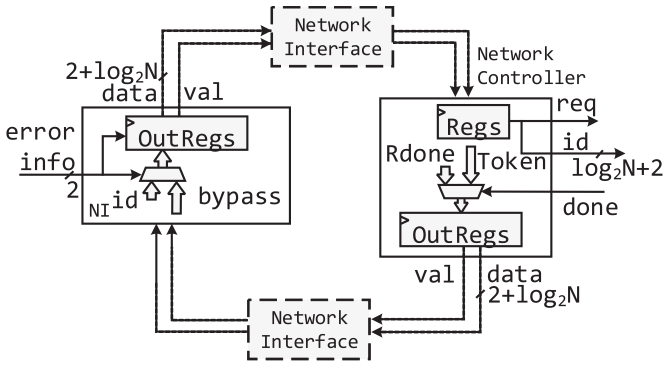

Fig. 4. The architecture of a token ring network.

```
logic location file *.ll

<bit offset> <frame addr> <offset> <Information>
Bit 19488835 0x0042021f 3107 Block=SLICE_X4Y48 Latch=AQ Net=voters[6]/status_bits[1]
```

Fig. 5. Extract of a Xilinx logic allocation file.

6. Fault emulation system

In this section, we outline the fault emulation system we implemented to assess the soft error vulnerability of the RCNs we studied.

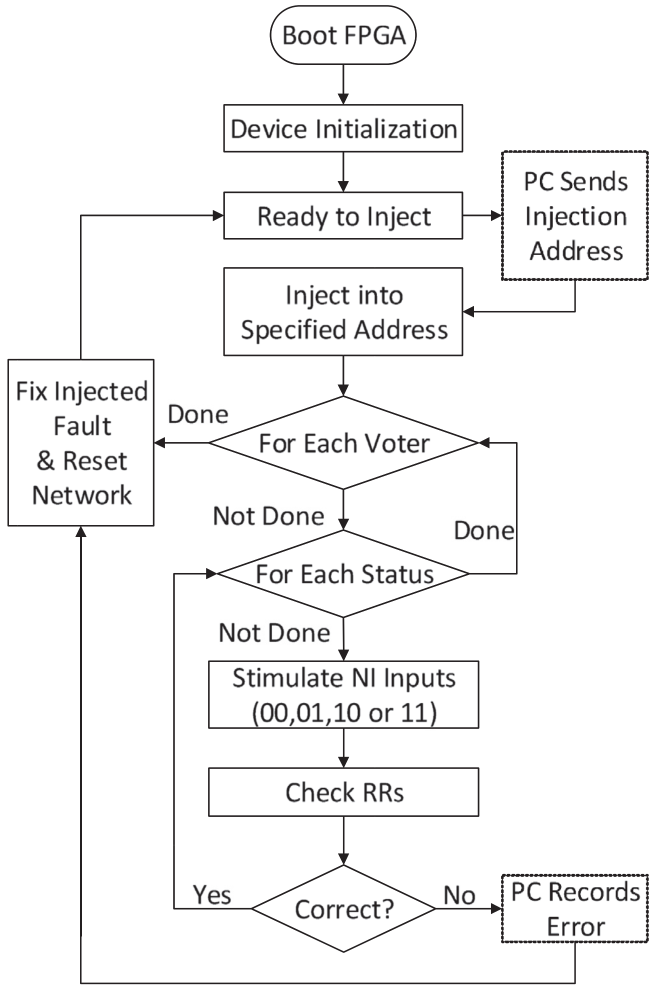

Fig. 6. Fault injection flowchart.

Fault emulation involves the use of hardware-based methods to artificially insert faults into FPGAs. A typical fault emulation system needs to provide an ability to access internal memory to inject a fault, an ability to stimulate and execute the circuits, an ability to determine output errors and an ability to clear errors [17].

Fig. 6 outlines our fault emulation procedure. We use the Xilinx AXI HWICAP IP for flipping configuration memory bits and a MicroBlaze processor to control this process. Once the system has been initialized, the MicroBlaze halts and waits for a fault injection address from a PC host. A uniformly distributed random configuration bit address is generated. The MicroBlaze reads the corresponding frame, flips the addressed bit and writes the frame back using the HWICAP to emulate an SEU. Note that we do not inject faults into either the MicroBlaze or the HWICAP in order to avoid their corruption through the fault injection campaign. Of the 18,300 configuration frames in the Artix-7 XC7A200TFBG-484 targeted in our study, 14,250 frames are contained in the region that represents the design under test.

Once a fault is inserted the circuit is tested using all possible input stimuli. In our case, there are four possible *error info* values that would normally be presented to a Network Interface (NI). An input stimulus is usually provided through external pins on the FPGA. However, we felt

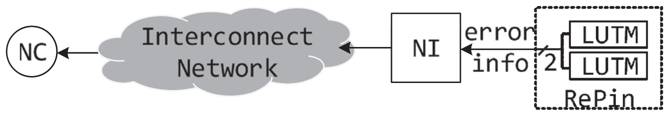

Fig. 7. RePin architecture for input stimulus.

that errors injected into the nets leading from the external pins to the NIs of the network would influence the results of our experiments. We therefore developed a different method for stimulating input values. As can be seen in Fig. 7, the input stimulus of each NI is realized through what we call a “RePin” architecture which is composed of two SLICEM LUT (LUTM) blocks configured as distributed RAM. Each LUTM provides a single bit of stimulus that can also be changed using the ICAP. Given the site number and logic locations, the positions of the LUTM bits can be obtained from the `*.ll` file as described in Section 5.

An essential step in the fault emulation procedure involves detecting system errors. The MicroBlaze processor checks the integrity of the design while the LUTMs are manipulated to simulate different *error info* signals arising from a voter. For each NI, we iterate through every possible combination of the *error info* signal (11, 00, 01 and 10) while holding the inputs to every other NI constant at 11, which signifies the “no error” condition. Whenever a new *error info* value is written, the MicroBlaze processor checks that the correct reconfiguration request (RR) is received. In the case of the soft networks, we wait for the maximum number of clock cycles required for the Network Controller (NC) to receive an RR, then the *req* and *module id* signals are read using the AXI GPIO interface. In the case of the ICAP-based RCN, the MicroBlaze processor utilizes the ICAP to read back the values of each NI’s status flip-flops in order to determine the RR. If the RR is as expected, we change the *error info* signal to the next value. When we have cycled through every possible status and there is no unexpected RR, we move on to the next NI. If an unexpected RR is received, an error report is sent to the PC.

The fault emulation tool must also remove the injected fault and return the circuit to a known functioning state before injecting the next fault. In our system, the injected fault is fixed by writing back the frame as it was before injection, all NI inputs are set to 11, the RCN is reset and the software returns to wait for a new fault injection address from the PC.

7. Reliability evaluation

In this section, we outline how we model the reliability of a non-replicated component, the reliability of a TMR component and the reliability of a complete FPGA-based system composed of both non-replicated and triplicated components. Our analysis is based on the number of critical bits per component for which we use the number of essential bits reported by the vendor’s tools as a worst case estimate. Whereas essential bits define the configuration of a user circuit on the target FPGA, the critical bits lead to a change in circuit behaviour if flipped.

Since without lengthy, exhaustive testing it is not possible to identify the critical bits, we pessimistically assume that the flip of a single essential bit leads to a module failure if the module is not triplicated. With this assumption, the module failure rate λ_m is given by the product of the bit error rate, λ_{bit}, and the number of essential bits in module m. Since all three modules are functionally identical, we also assume that the three modules of a TMR component have the same failure rate λ_m.

Using SPENVIS [18], we calculated that the configuration memory of Xilinx 7 Series FPGAs, specifically the Kintex-7 family will upset at a peak rate of $\lambda_{bit} = 2.7 \times 10^{-10}$ upsets/bit/s in equatorial geosynchronous orbit. To determine this upset rate, we used the peak 5-minute average flux of the CREME-96 model [19] and, as is typically done, assumed the presence of 2.54 mm of aluminium shielding. The cross section of the Kintex-7 family was obtained from [20]. In this paper, we use this estimate as a “relatively high radiation level” for our analysis.

As is usual [21], we assume that module reliability decreases exponentially over time t as expressed by the function:

$$R_m(t) = e^{-\lambda_m t}, \tag{3}$$

whereby the reliability, $R_m(t)$, of a module at time t denotes the probability that the module operates without any failure in the interval [0, t].

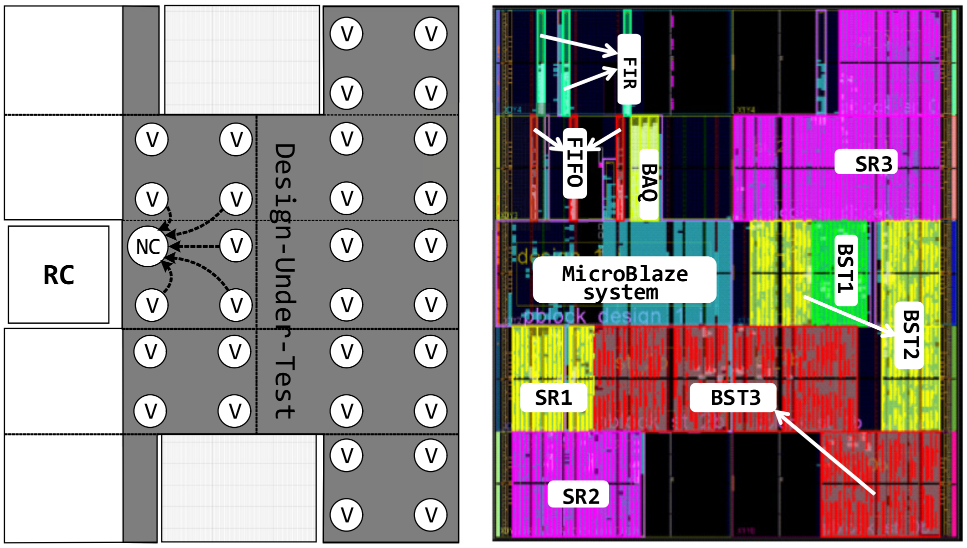

Fig. 8. (a) Synthetic layout of a 31-voter design and (b) RUSH floorplan.

When module m is triplicated, its reliability function becomes:

$$R_m^{TMR}(t) = 3R_m^2(t) - 2R_m^3(t). \tag{4}$$

In order to achieve higher reliability, for a given SEU rate, we employ TMR with MER. The reliability function is then given by [22]:

$$R_{m,r}^{TMR}(t) = \frac{e^{-\frac{1}{2}(at)}\left(a\sinh\left(\frac{bt}{2}\right) + b\cosh\left(\frac{bt}{2}\right)\right)}{b}, \tag{5}$$

where $a = 5\lambda_m + \mu_m$, $b = \sqrt{\lambda_m^2 + 10\lambda_m\mu_m + \mu_m^2}$.

The term μ_m denotes the repair rate of a module, which is the reciprocal of the time needed to recover the faulty module:

$$\mu_m = \frac{1}{t_{repair}} = \frac{1}{t_d + t_c + t_{sync}} \approx \frac{1}{t_d + t_c}, \tag{6}$$

where t_d denotes the average error detection time, t_c denotes the error correction time and t_{sync} denotes the synchronization time, which we omit in our case study because it normally only accounts for a small fraction of the recovery time.

Note that t_d depends on the method used to detect errors and corresponds in our case to the average latencies that were derived in Eqs. (1) and (2), whereas t_c, which depends on parameters of the target system and the size of the module, is given by the number of 32-bit words per frame, the number of frames in the given TMR module and the ICAP write throughput.

The reliability of an FPGA-based system composed of N TMR components that use MER to recover from configuration memory errors and an RCN for aggregating reconfiguration requests can be derived as follows. We model the reliability of the RCN $R_{RCN}(t)$ using Eqs. (3, 4 or Eqs 5). Respectively, the reliability of each TMR component $R_{i,r}^{TMR}(t)$ in the system is modelled using Eq. (5). Finally, the reliability of the system is given by the product of the reliability of each individual component, namely the RCN and the N TMR components [21]:

$$R_s^{TMR}(t) = R_{RCN}(t)\prod_{i=1}^{N} R_{i,r}^{TMR}(t). \tag{7}$$

In this derivation, it is assumed that failures follow a Poison distribution and the occurrence of errors in modules or components are statistically independent and uncorrelated. Note that Eq. (7) holds true only if $\mu \gg \lambda$, which ensures repairs are completed independently [21]. Moreover, since the main objective of this paper is to evaluate the impact of various RCN architectures on the total reliability of FPGA-based designs that incorporate MER, we omit inclusion of the reconfiguration controller and the voters in our reliability analysis.

8. Experiments and results

In this section we evaluate the performance of the networks presented in Section 4, in terms of resource utilization, latency, operating frequency, power consumption and soft-error vulnerability. All networks were implemented on a Xilinx Artix-7 XC7A200TFBG484-1 FPGA, as hosted on the RUSH experiment board [13], using the vendor's Vivado 2014.4 implementation tools with default settings. The comparison of the networks is based on data obtained from the implementation tools and also on fault-injection experiments.

8.1. Experiments

As mentioned in Section 4, an RCN consists of NIs, a central NC and the interconnection network between them. In our experiments the same voter interface and RC designs were used in each experiment irrespective of the RCN type being tested. The same NI and NC locations were also used for all RCN designs. In a first experiment we studied "synthetic" layouts in which the TMR components, their voters, and thus the NIs were assumed to be distributed in a checkerboard pattern across the majority of the device area. Moreover, the NIs and the NC were always located in partitions that utilized the same FPGA resources irrespective of the RCN topology. To obtain resource utilization and performance results, we initially implemented designs that only contained the components of the RCNs being tested and constrained the implementation tools to prevent optimizations across the port interfaces of the NIs and the NC. To perform the fault injection experiments, we added a MicroBlaze-based RC for injecting faults and distributed RAM-based test vectors to each of the RCNs we tested. We tested each RCN type for networks comprising 7, 15 and 31 voters. The synthetic layout of a 31-voter design (in this case for testing the star network topology) is shown in Fig. 8(a), in which the design under test into which faults were injected is depicted as the shaded region to the right of the RC.

In a second experiment, we investigated the utilization and performance of each RCN when used to collect reconfiguration requests for the RUSH board [13]. For this case study, we implemented the four network types with the 9 TMR components comprising the RUSH experiment. These components include a single MAC-based 21-tap *Finite Impulse Response* (FIR) filter with 16-bit signal width, an 8-to-3-bit *Block Adaptive Quantizer* (BAQ), an 8096-word deep 32-bit *FIFO*, three 32-bit *Shift Registers* (SRs) having different lengths and a range of

Table 1
Results of mapping four RCNs to Xilinx Artix-7 XC7A200TFBG-484.

Type	ICAP					STAR				BUS				RING			
Layout	L1			L1*	L2	L1			L2	L1			L2	L1			L2
# NIs	7	15	31	31	9	7	15	31	9	7	15	31	9	7	15	31	9
Slices	7	15	31	31	9	12	29	50	18	21	33	60	21	30	50	141	35
LUTs	0	0	0	0	0	14	27	30	16	28	50	108	33	54	130	279	87
FFs	14	30	62	62	18	26	44	77	32	35	61	110	43	61	134	295	87
PIPs	440	889	1770	1858	557	1101	1996	3513	1,243	1341	2553	4625	1729	2057	3894	7986	2724
SMs	38	62	102	181	55	277	453	792	274	351	616	1074	466	426	496	861	426
Freq. (MHz)			100			112	109	107	126	109	107	104	114	132	203	186	145
Clocks / Hop	230					2				2				1			
# hops	7	15	31	8	9	7	15	31	9	7	15	31	9	8	16	32	10
Latency (us)	8.05	17.25	35.65	9.20	300	0.06	0.14	0.29	0.5	0.06	0.14	0.30	0.5	0.07	0.08	0.18	0.5
Static (mW)	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138
Dynamic (mW)	3	4	5	5	3	4	7	7	4	6	7	9	5	4	5	8	4
Ess. bits (Kb)	3.42	5.6	10.4	13.7	4.1	9.4	15.8	26.0	10.1	11.9	20.1	38.6	14.9	18.3	33.7	69.4	24.3

L1: Synthetic layout.
L2: RUSH layout.
L1*: optimized ICAP layout.

combinational logic between the stages and three 32-bit *Binary Search Trees* (BSTs) of different heights and a range of combinational logic at each node. A MicroBlaze processor was used to implement the RC and the AXI HWICAP IP was used to reconfigure faulty modules. The layout of this system is depicted in Fig. 8(b).

8.2. Results

8.2.1. Implementation results

Table 1 presents information extracted from the vendor's implementation tools. The results are listed according to the resource utilization of the design; the dynamic power consumption and the number of essential bits follow the same pattern. In contrast, the vendor's power analysis tools reported the same amount of static power consumption for all RCN designs. However, given that the RCN designs utilized less than 0.2% of the total FPGA resources on average, we believe that the contribution of the RCN to the total static power consumption of the FPGA is negligible, and due to this we have obtained the same result for all designs.

It can be seen that the ICAP-based RCN was realized with the fewest resources compared to the other RCN architectures. This is primarily because the ICAP NIs are implemented with just two Flip-Flops (FFs) and a small amount of support logic being mapped to Look-Up Tables (LUTs). As expected, the number of Programmable Interconnection Points (PIPs) and Switch Matrices (SMs) used by the ICAP approach is significantly lower than for the other approaches. As a consequence, the ICAP-based RCN has on average 2.7, 3.6 and 6.0 times fewer essential bits than the synthetic layouts of the star, bus and ring networks respectively. However, the ICAP-based RCN suffers from high network latency. It requires two to three orders of magnitude more time than the other RCNs to transfer reconfiguration requests to the NC. In contrast, the ring has the lowest latency, since it can achieve a higher operating frequency and only needs 1 clock cycle per node hop. We used Eqs. (1) and (2) to calculate the latency for each RCN. The latency of the ICAP approach is on average over 175 times that of the ring and the latency of the star and bus networks was about 1.4 times that of the ring for the synthetic layouts.

We investigated an optimization of the ICAP RCN that entails constraining the registers of those groups of NIs that are located within each clock region. These registers are forced to be placed into a single configuration frame so that they can be accessed in a single frame read. With reference to Fig. 8(a), which depicts 4 voters per clock region (the 10 grey rectangles), this optimization resulted in the creation of horizontal wires leading from each voter to a frame that was centrally located in each clock region. Instead of requiring 31 separate frame reads to check all voters, this approach reduced the number of frame reads needed to 8 in total — one for each clock region used by the design. The results of this implementation are reported in Table 1 in the ICAP column headed L1*. As can be seen, this optimization reduced the latency of the ICAP approach by a factor of 4 while increasing the number of essential bits used over the unoptimized 31-voter ICAP design by 32%.

8.2.2. Fault injection results

We implemented the fault emulation system described in Section 6 to conduct fault injection experiments for the synthetic layouts of each of the four RCN types. Table 3 tabulates the average number of errors we found after five trials of one million fault injections. These results demonstrate that the ICAP-based RCN is more reliable than the other approaches. Additionally, the number of errors that occur in each RCN is directly proportional to the number of voters and thus the number of essential bits per design.

8.2.3. RUSH case study results

Table 2 presents the number of essential bits (n_e), the failure rate of each module assuming $\lambda_{bit} = 2.7 \times 10^{-10}$ upsets/bit/s, the number of frames (n_f) and the correction time (t_c) of each TMR module. Note that in our design, since we were using the AXI HWICAP, the ICAP throughput using the MicroBlaze was limited to 10 MB/s, considerably less than the maximum possible throughput of 400 MB/s. The reduction in ICAP bandwidth also affected the latency for checking a voter using the ICAP to 60 us per voter, and we therefore observed a much higher network latency.

Fig. 9 plots the system reliability for each RCN type and the 9 RUSH application circuits using Eq. (7) against the reliability of a blind scrub implemented on the same system. The MicroBlaze RC and off-chip flash configuration storage used by the RUSH system supports a random FPGA configuration frame read latency of 60 us and a sustained frame write period of 18 us per frame. Blind scrubbing, which entails rewriting each configuration frame of the device, therefore takes 330 ms on the Artix-7 XC7A200TFBG-484 used, and errors are recovered by scrubbing after 165 ms on average. Please note that in Fig. 9, the scrub

Table 2
Results of mapping 9 TMR components to Xilinx Artix-7 XC7A200TFBG-484.

Design	Utilization				Essential Bit	Failure rate (λ_m)	n_f	t_c (ms)
	LUTs	FFs	DSP	BRAM	(n_e)	(upsets/s/module)		MicroBlaze
FIR	33 (0.02%)	16 (0.01%)	1 (0.13%)	–	12,042 (0.02%)	3.25×10^{-6}	65	1.2
FIFO	72 (0.05%)	111 (0.04%)	–	7.5 (2.05%)	41,842 (0.07%)	1.13×10^{-5}	192	3.5
BAQ	305 (0.22%)	197 (0.07%)	–	–	48,963 (0.08%)	1.32×10^{-5}	73	1.3
BST1	1396 (1.04%)	2519 (0.94%)	–	–	281,604 (0.46%)	7.60×10^{-5}	145	2.6
SR1	1619 (1.20%)	3273 (1.22%)	–	–	285,914 (0.46%)	7.72×10^{-5}	378	6.8
SR2	2630 (1.96%)	5499 (2.05%)	20 (2.70%)	–	515,904 (0.84%)	1.39×10^{-5}	474	8.5
BST2	3779 (2.84%)	6198 (2.32%)	31 (4.18%)	–	793,534 (1.30%)	2.14×10^{-4}	610	11.0
SR3	7022 (5.24%)	14,573 (5.44%)	40 (5.40%)	–	1,403,647 (2.30%)	3.79×10^{-4}	1090	19.6
BST3	9126 (6.82%)	12,214 (4.56%)	31 (4.18%)	–	1,833,235 (3.00%)	4.95×10^{-4}	1483	26.7

Table 3
Fault injection results.

Type	ICAP		STAR		BUS		RING	
# voters	Avg	σ	Avg	σ	Avg	σ	Avg	σ
7	7.0	1.5	8.2	2.3	16.8	2.1	51.0	4.7
15	8.2	3.5	17.0	3.0	36.6	5.0	122.1	16.7
31	20.7	1.4	38.6	4.6	78.6	7.9	213.4	27.3

Avg: Average number of observable errors.
σ: Standard deviation.

plots only account for the 9 application components; they specifically exclude an RCN component, which is not needed for blind scrubbing.

Fig. 9(a) assumes the four RCNs are implemented as single, non-replicated components. While the ICAP RCN results in the best reliability for MER, all 4 RCNs weigh down the reliability of the system because they are single points of failure.

Fig. 9(b) assumes the RCNs are implemented as triplicated components, but that errors that occur in this component are not repaired. Some limited error mitigation is therefore in place. Only the ICAP outperforms scrubbing over the time period shown. However, eventually (when $t > 120,000$ s) even this approach succumbs to errors that remain unrepaired and scrubbing once again dominates.

In Fig. 9(c) we assume that the device is partially reconfigured in its entirety when an error in the triplicated RCN component is detected. This error recovery period is longer than desired, but the approach ensures any error in the network is corrected. Despite the long recovery time (equivalent to reconfiguring the complete device), the reliability is not significantly affected because errors occur infrequently in the relatively small RCN components.

9. Concluding remarks and future work

In this paper, we have compared four RCN types in terms of reliability, scalability, resource utilization, power consumption and sensitivity to configuration memory errors. The utilization and performance of these RCNs were assessed for networks with 7, 15 and 31 voters. The results demonstrate that the ICAP-based readback approach, which uses the built-in reconfiguration mechanism available in FPGAs, requires the least resources of those networks studied.

The results of a case study that was implemented on the RUSH payload and of fault injection testing indicate that the ICAP-based readback approach has the highest system reliability despite having a relatively high latency. This higher latency may not be too problematic except when radiation levels become much higher than the high rate assumed in our work. We have shown that the latency of the ICAP approach can be reduced by clustering the registers that are to be read from one clock region into a single frame. This optimization does not have a significantly impact on the resource utilization. We have also determined that for the reliability of MER to be competitive with scrubbing in a real system, the RCN must also be triplicated and repaired when errors affect it.

One direction for further study is to consider the order in which TMR components are checked. Further work is also envisaged to derive more comprehensive reliability models for complete FPGA-based TMR systems with MER. This work is just the first step in this direction.

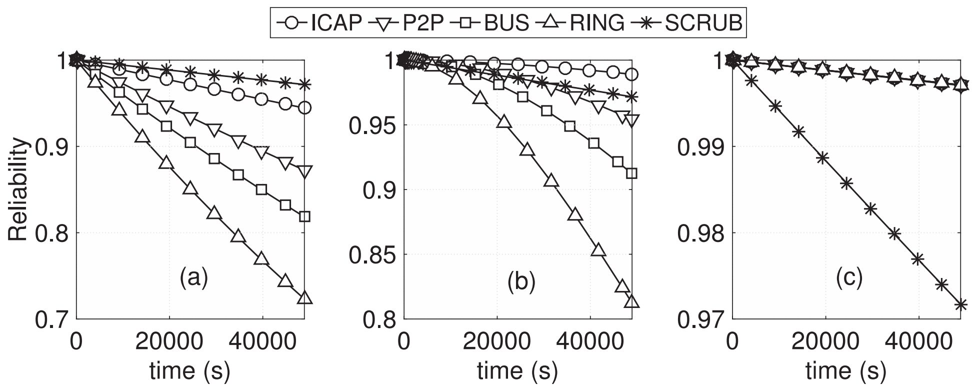

Fig. 9. (a) Unprotected RCN (b) TMR triplicated (c) TMR triplicated with recovery.

Acknowledgement

This research was supported in part by the Australian Research Council's Linkage (LP140100328) and Discovery (DP150103866) Projects funding schemes.

References

[1] F. Siegle, T. Vladimirova, J. Ilstad, O. Emam, Mitigation of radiation effects in SRAM-based FPGAs for space applications, ACM Comput. Surv. 47 (2) (2015) 37:1–37:34, http://dx.doi.org/10.1145/2671181.
[2] I. Herrera-Alzu, M. Lopez-Vallejo, Design techniques for Xilinx Virtex FPGAconfiguration memory scrubbers, IEEE Trans. Nuclear Sci. 60 (1) (2013) 376–385, http://dx.doi.org/10.1109/TNS.2012.2231881.
[3] J. von Neumann, Probabilistic logics and the synthesis of reliable organisms from unreliable components, Autom. Stud. (1956) 43–98.
[4] R.E. Lyons, W. Vanderkulk, The use of triple-modular redundancy to improve computer reliability, IBM J. Res. Dev. 6 (2) (1962) 200–209, http://dx.doi.org/10.1147/rd.62.0200.
[5] C. Carmichael, M. Caffrey, A. Salazar, Correcting Single Event Upsets Through Virtex Partial Configuration, Xilinx, 2000. XAPP216
[6] C. Bolchini, A. Miele, C. Sandionigi, A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs, IEEE Trans. Comput. 60 (12) (2011) 1744–1758, http://dx.doi.org/10.1109/TC.2010.281.
[7] M. Straka, J. Kastil, Z. Kotasek, L. Miculka, Fault tolerant system design and SEU injection based testing, Microprocess. Microsyst. 37 (2) (2013) 155–173, http://dx.doi.org/10.1016/j.micpro.2012.09.006. Digital System Safety and Security
[8] E. Cetin, O. Diessel, L. Gong, V. Lai, Reconfiguration network design for SEU recovery in FPGAs, IEEE International Symposium on Circuits and Systems (ISCAS), (2014), pp. 1524–1527, http://dx.doi.org/10.1109/ISCAS.2014.6865437.
[9] J. Tonfat, F.L. Kastensmidt, P. Rech, R. Reis, H.M. Quinn, Analyzing the effectiveness of a frame-level redundancy scrubbing technique for SRAM-based FPGAs, IEEE Trans. Nuclear Sci. 62 (6) (2015) 3080–3087, http://dx.doi.org/10.1109/TNS.2015.2489601.
[10] PG036: Vivado design suite soft error mitigation controller v4.1, 2015.
[11] B. Navas, J. berg, I. Sander, The upset-fault-observer: a concept for self-healing adaptive fault tolerance, NASA/ESA Conference on Adaptive Hardware and Systems (AHS), (2014), pp. 89–96, http://dx.doi.org/10.1109/AHS.2014.6880163.
[12] F. Veljkovi, T. Riesgo, E. de la Torre, Adaptive reconfigurable voting for enhanced reliability in medium-grained fault tolerant architectures, NASA/ESA Conference on Adaptive Hardware and Systems (AHS), (2015), pp. 1–8, http://dx.doi.org/10.1109/AHS.2015.7231165.
[13] E. Cetin, O. Diessel, T. Li, J.A. Ambrose, T. Fisk, S. Parameswaran, A.G. Dempster, Overview and Investigation of SEU Detection and Recovery Approaches for FPGA-Based Heterogeneous Systems, Springer International Publishing, Cham, 2016, pp. 33–46, http://dx.doi.org/10.1007/978-3-319-14352-1_3.
[14] D. Agiakatsikas, N.T.H. Nguyen, Z. Zhao, T. Wu, E. Cetin, O. Diessel, L. Gong, Reconfiguration control networks for TMR systems with module-based recovery, IEEE International Symposium on Field-Programmable Custom Computing Machines, (2016), pp. 88–91, http://dx.doi.org/10.1109/FCCM.2016.30.
[15] UG470: Xilinx 7 series FPGAs configuration user guide v1.10, 2015.
[16] A. Sari, M. Psarakis, Scrubbing-based SEU mitigation approach for systems-on-programmable-chips, International Conference on Field-Programmable Technology (FPT), (2011), pp. 1–8, http://dx.doi.org/10.1109/FPT.2011.6132703.
[17] H. Quinn, M. Wirthlin, Validation techniques for fault emulation of SRAM-based FPGAs, IEEE Trans. Nuclear Sci. 62 (4) (2015) 1487–1500, http://dx.doi.org/10.1109/TNS.2015.2456101.
[18] D. Heynderickx, B. Quaghebeur, E. Speelman, E. Daly, ESAs space environment information system (SPENVIS): a WWW interface to models of the space environment and its effects, in AIAA, 371, 2000.
[19] A. Tylka, J. Adams, P. Boberg, B. Brownstein, W. Dietrich, E. Flueckiger, E. Petersen, M. Shea, D. Smart, E. Smith, CREME96: a revision of the cosmic ray effects on micro-electronics code, IEEE Trans. Nuclear Sci. 44 (6) (1997) 2150–2160, http://dx.doi.org/10.1109/23.659030.
[20] D. Hiemstra, V. Kirischian, Single event upset characterization of the Kintex-7 Field Programmable Gate Array using proton irradiation, REDW, (2014), pp. 1–4, http://dx.doi.org/10.1109/REDW.2014.7004593.
[21] M.L. Shooman, Reliability of Computer Systems and Networks: Fault Tolerance, Analysis and Design, John Wiley & Sons, Inc., New York, NY, USA, 2002.
[22] D. McMurtrey, K.S. Morgan, B. Pratt, M.J. Wirthlin, Estimating TMR reliability on FPGAs using Markov models, 2008, http://scholarsarchive.byu.edu/facpub/149.

Nguyen T. H. Nguyen is a Lecturer in the Faculty of Computer Science & Engineering, at the Ho Chi Minh City University of Technology. He received B.E., M.S. and Ph.D. degrees from the Ho Chi Minh City University of Technology, Vietnam, Kanazawa University, Japan and UNSW Sydney, Australia respectively. He is interested in methodologies for design and analysis of computing system with a focus on fault tolerant techniques and reliability-related analyses.

Dimitris Agiakatsikas is a Ph.D student at the University of New South Wales, Australia. He received a B.Sc. in Electronics from the Technological Educational Institute of Athens, Greece and a M.Sc. in technology of Embedded Systems from the University of Piraeus, Greece. Before commencing his Ph.D studies, he was an Electronics Engineer at the National Observatory of Athens, Greece. His research interests include fault-tolerant computing, computer-aided design for fault-tolerant FPGA-based systems and dependability modelling.

Ediz Cetin is a Senior Lecturer in the School of Engineering, at Macquarie University, Sydney, Australia. He received his B.Eng. (Hons) degree in Control and Computer Engineering and Ph.D. degree in Unsupervised Adaptive Signal Processing for Wireless Receivers from the University of Westminster, London, United Kingdom. His research interests encompass interference detection and localization, fault-tolerant reconfigurable circuits for space applications, adaptive techniques for RF impairment mitigation for communications and Global Navigation Satellite Systems (GNSS) receivers and design and low-power implementation of digital circuits.

Oliver Diessel is an Associate Professor in the School of Computer Science & Engineering, at the University of New South Wales in Australia. He gained B.E., B. Math., and Ph.D. degrees from the University of Newcastle, Australia. His research interests encompass the design and application of dynamically reconfigurable systems and technology, including modelling, design methods, and run-time support.